Guarded Matching Logic is Decidable

Nishant Rodrigues
Xiaohong Chen
Grigore Rosu
nishant2@illinois.edu
xc3@illinois.edu
grosu@illinois.edu

Abstract

Matching logic is the logical foundation of the $\mathbb{K}$ language framework, where formal semantics of programming languages can be defined and language tools are automatically generated from the semantics. To bring more automation to $\mathbb{K}$ and its formal reasoning tools, we need to study decision procedures for matching logic.

In this paper, we present three increasingly powerful decidable fragments of matching logic. The first, "modal matching logic", does not allow fixedpoints or quantification and is akin to multimodal polyadic modal logic. The second, "quantifier-free matching logic", extends this to allow fixedpoints. Finally, "guarded matching logic", allows both fixedpoints and quantification, albiet in a carefully restricted form. We prove that each of these fragments are decidabile, and several existing decidability results for modal μ-calculus, infinite- and finite-trace linear temporal logic, computation tree logic, and dynamic logic are all corollaries.

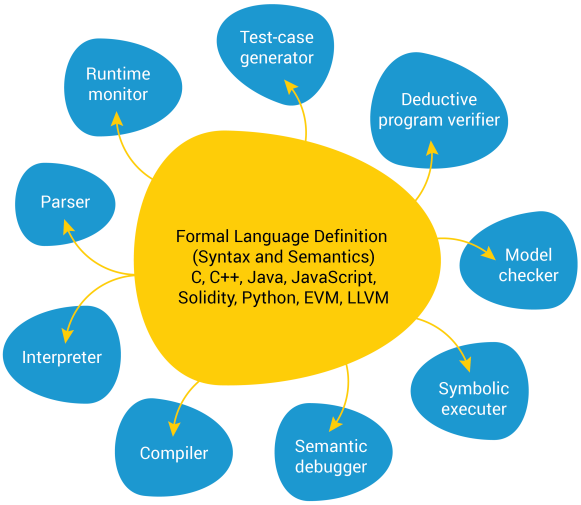
1 Motivation

The proliferation of new programming languages in recent times has made clear the need for an *ideal language framework*. Each new language is tailored to its domain and yet, rightfully, demands sophisticated tooling for debugging, model checking and program verification. Implementing each of these tools individually for each language is redundant and cost-prohibitive. As shown in Figure 1a, an ideal language framework allows language designers to specify the formal syntax and semantics of their language, and have all tools (such as parsers, interpreters, compilers, and even model checkers and deductive verifiers) automatically generated by the framework. Besides reducing the duplication of effort between developing such tools multiple times for different languages, it ensures that each tool uses the same internal model for the language. For example, since the program verifier and interpreter are defined from the same "source of truth", running functional tests against the interpreter gives us increased confidence in the correctness of the deductive prover.

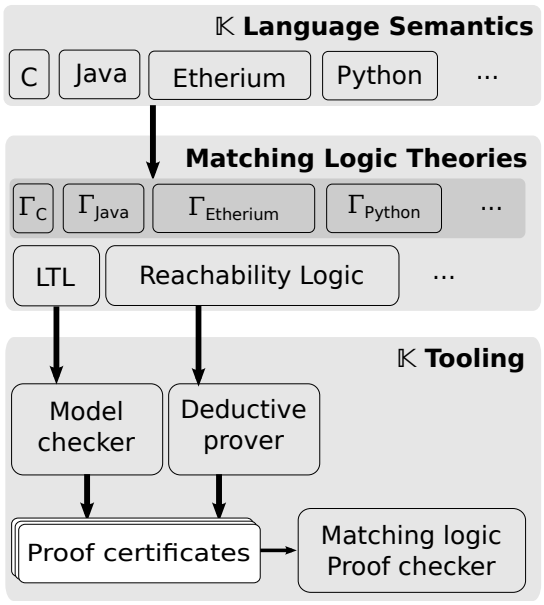
The $\mathbb{K}$ framework (kframework.org) pursues this vision. $\mathbb{K}$ provides an intuitive meta-language with which language designers may define the formal semantics of their programming language as a transition system. From this formal semantics, the framework generates parsers, interpreters, deductive verifiers [Ştefănescu et al. 2016; Roşu 2017a], program equivalence checkers [Kasampalis et al. 2021], among others. Diverse and complex programming languages have been specified in $\mathbb{K}$ including C [Hathhorn et al. 2015], Java [Bogdanas and Roşu 2015], JavaScript [Park et al. 2015], Ethereum virtual machine [Hildenbrandt et al. 2018] and x86 assembly [Dasgupta et al. 2019]. The implementation of complex languages such as C and Java show that this approach is not just an academic dream anymore. The commercial success of verification tools built using this approach (runtimeverifcation.com) show that these tools are practical, valuable and in-demand.

$\mathbb{K}$ needs a firm logical foundation in order to provide sound and powerful formal verification tools. Matching logic [Chen et al. 2021b; Chen and Roşu 2019; Roşu 2017b] provides this foundation. As shown in Figure 1b, every $\mathbb{K}$ semantic definition of a language L yields a corresponding matching logic theory Γ_L, and every language task (such as executing a program or verifying a property) conducted by $\mathbb{K}$ is characterized by a matching logic proof $\Gamma_L \vdash \varphi_{task}$, where φ_{task} is the formal specification of the task in matching logic. These language tasks range from running a program (i.e. does there exist a terminating execution trace), to proving reachability claims. If these tools emit proof certificates, they may be checked with the matching logic proof checker [Chen et al. 2021a].

Matching logic provides this foundation by creating a unifying logic, or *lingua franca*, for formal verification. Using constructs for building terms, first-order quantification and fixedpoint/inductive reasoning, it allows capturing the language-semantics-as-a-theory produced by $\mathbb{K}$ while simultaneously providing an environment for carrying out proofs about programs and languages. Because it preserves and respects the original syntactic and semantic structures, such as programs, continuations, heaps and stacks, language semantics may be captured in a compact and modular way; in contrast to approaches that translate these, sometimes awkwardly, to a fixed set of constructs. Many formalisms important to verification have been embedded in matching logic (LTL, CTL, separation logic, reachability logic, etc) [Chen and Roşu 2019; Roşu 2017b]. These embeddings combined



(a) Vision of an ideal language framework. All language tools are generated from one formal semantics.



(b) Matching logic as the logical foundation of 𝕂.

Figure 1. Motivation for 𝕂 and matching logic.

with the language theories may be used to define the various language tasks described above.

𝕂's tools are best-effort checking for the validity of the corresponding entailments. Currently, this is done through ad-hoc reasoning developed on an as-needed basis and translation to SMT-LIB2 [Barrett et al. 2010] for dispatch to the Z3 solver [De Moura and Bjørner 2008]. This leads to quite a few deficiencies — limited support for induction, users need to spell out many lemmas and simplifications, caching and optimization are at the mercy of what Z3's incremental interface will accept.

Our grand vision is to develop a matching logic solver, systematically and methodically, to alleviate these problems. Solvers for first-order logic are typically constructed around DPLL [Davis and Putnam 1960], an algorithm for checking the satisfiability of propositional logic formulae. A first-order formula is transformed into a propositional "skeleton" by replacing atoms with propositional variables. The DPLL algorithm then produces solutions to this skeleton – truth assignments to each of the introduced propositional variables. If any of these solutions are consistent with the atoms from the original formula, then the entire first-order formula is satisfiable. Unfortunately, DPLL cannot directly be used in the same way as the core for matching logic automated proving because matching logic formulae cannot be reduced to a propositional skeleton. This is because matching logic patterns are interpreted as the set of elements they match, unlike propositional variables which are two-valued (true or false). Similarly to modal logics, translation to first-order logic or other logics is not desired in general, because and additional level of complexity is added, such as new quantifiers, and because many of its nice properties can be lost in translation.

In this paper we propose three increasingly powerful decidable fragments of matching logic. The final fragment, called *guarded matching logic*, allows both fixedpoints and quantification. A decision procedure for this fragment may be used as the core of a matching logic solver, analogously to DPLL. Since it allows axioms, decidability for many embedded logics, including modal μ-calculus, infinite- and finite-trace linear temporal logic, computation tree logic, and dynamic logic, are subsumed into this result.

2 The development of Guarded Logics

Inspired by the robust decidablity properties of modal logic, guarded logics were created as a means of "taming" a logic, i.e. of restricting a logic so that it becomes decidable. This is done through syntactic restrictions on quantification. By "decidable", we refer to the decidability of the satisfiability problem:

> Given an axiom set Γ and a formula φ, does there exist a model M and an assignment for the free variables in φ to elements in M, such that M validates Γ and φ.

This problem, of finding an algorithm for checking if a formula is satisfiable, was originally posed by Hilbert and Ackermann as a fundamental challenge to the mathematical community in 1928 [Hilbert and Ackermann 1930]. Alas, this

holy grail of mathematics was proven unattainable for arbitrary mathematical statements by Church [Church 1936] and Turing [Turing 1937] in 1936. This did not cause the problem to go away – it merely turned into one of classification: for what classes of mathematical formulae may we have such an algorithm?

One way of defining such classes of formulae is by placing syntactic constraints on formulae and axioms over which an algorithm may safely operate. We call these syntactic criteria "fragments" of a logic. Early results focused on the quantifier prefix classes (restricting the number and order in which quantifiers may be used) and vocabulary (limiting the number and arity of relation and function symbols).

The study of modal logic, a decidable logic commonly used in program verification, among other fields, led to several important results. Modal logic extends propositional logic with two constructs: $\langle\alpha\rangle\varphi$ and $[\alpha]\varphi$, where α is drawn from a set of "actions" or "modalities". $\langle\alpha\rangle\varphi$ means that φ holds for some α-successor of the current state and $[\alpha]\varphi$ means that φ holds for all α-successors. While modal logic is a fairly simple logic, several powerful extensions, such as with transitive closure operators and least and greatest fixedpoints preserve its decidability.

Initially, it was thought that modal logic's decidability was due to it being a subset of FO^2, the fragment of first-order logic that allows only two variables (known to be decidable [Mortimer 1975]), under the following translation:

$$\begin{aligned}\mathsf{Tr}(\varphi) &\mapsto \mathsf{Tr}(x,\varphi)\\ \mathsf{Tr}(x,\langle\alpha\rangle\varphi) &\mapsto \exists y.\, R_\alpha(x,y) \wedge \mathsf{Tr}(y,\varphi)\\ \mathsf{Tr}(x,[\alpha]\varphi) &\mapsto \forall y.\, R_\alpha(x,y) \rightarrow \mathsf{Tr}(y,\varphi)\end{aligned}$$

where x and y are fresh variables, R_α is a relational symbol associated with action α. However, this explanation wasn't very satisfactory – unlike with modal logic, extension with fixedpoints or transitive closure caused it to become undecidable.

[Vardi 1997] shows that the reason for this "robust" decidability was that its models have the "tree-model property", and that this property leads to automata based decision procedures. With this insight, fragments that preserved decidability under such extensions were identified. The *guarded fragment* of first-order logic defined in [Andréka et al. 1998] allows quantification over an arbitary number of variables so long as it is in the form:

$$\begin{aligned}&\exists y.\, \alpha(x,y) \wedge \psi(x,y) \quad \text{or,}\\ &\forall y.\, \alpha(x,y) \rightarrow \psi(x,y)\end{aligned}$$

where $\alpha(x,y)$ is a propositional atom, called a "guard" that mentions all free variables of ψ. This has since been generalized in two directions.

First, to allow more general guards. In *loosely guarded* first-order logic presented in [Hodkinson 2002], guards are allowed to be conjunctions of atoms, rather than just single atoms. *Packed logic* extends this further, allowing even existentials to occur in guards. In the *clique guarded* fragment of first-order logic [Grädel 2002], quantification is semantically restricted to cliques within the Gaifman graph of models.

Second, to allow fixedpoints: guarded fixedpoint logic, loosely guarded fixedpoint logic [Gradel and Walukiewicz 1999], and clique-guarded fixedpoint logic [Grädel 2002]. extend the corresponding guarded logics to allow fixedpoints constructs. An interesting property of guarded fixedpoint logics, is that despite being decidable, they admit "infinity axioms" – axioms that are satisfiable only in infinite models.

3 Overview

Matching logic as a whole, is undecidable since it is subsumes first-order logic [Roşu 2017b]. So, we must look for decidable fragments of matching logic. In this paper, we present three increasingly powerful fragments of matching logic that are decidable.

The first fragment, which we refer to as the "modal" fragment of matching logic, may be viewed as a polyadic multimodal variant of modal logic. This fragment allows neither quantification nor fixedpoints. We do this by showing that this fragment has the "small model property" – every satisfiable pattern has a model with size bound by a function on the size of the pattern – allowing us to exhaustively search for a model. This is done by producing a bounded "filtered model" from arbitary models by grouping elements into equivalence classes over satisfied sub-patterns of the original pattern, and showing that satisfiability is equivalent in this model.

The next fragment, quantifier-free matching logic, allows fixedpoints but no quantiification. It is proved decidabile following a similar approach to that in [Niwiński and Walukiewicz 1996]. This fragment enables us to capture the decidability of μ-calculus, and propositional dynamic logic as corollaries.

Finally, we present a more general fragment that allows guarded quantification, called "guarded matching logic". This result is proved through an satisfiability-preserving translation of matching logic patterns to fixedpoint logic, and showing that the image of this fragment lies in the loosely guarded fragment of fixedpoint logic, a decidable fragment. Since quantification enables the capture of axioms and constrain our models, this decidability holds not only in the case of empty theories, but also for non-empty theories when their axioms fall into this fragment. The fragment is closed under negation, implying that validity is also decidable. This allows us to derive several decidability results for matching logic theories. These results are summarized in Figure 2.

We organize the main results of our paper as follows:

- In **Section 4** and **Section 5**, we present a brief introduction to matching logic and fixedpoint logic ($\mathsf{FO}_{\mathsf{lfp}}$).
- In **Section ??**, we present a proof for the decidability of the fragment modal of matching logic, via the small-model property.

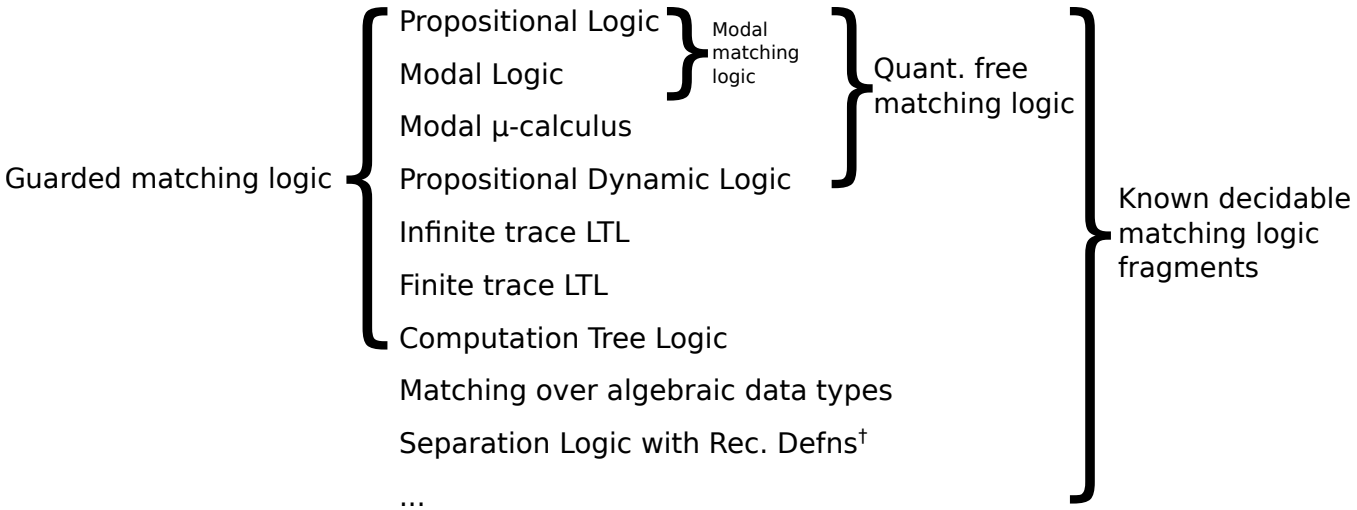


Figure 2. Many logics encoded in matching logic have existing decidability results. With guarded matching logic, we subsume many of these results through a simple syntactic check rather than a likely cumbersome proof.
†: We refer to the bounded tree width fragment of SLRD [Iosif et al. 2013].

- In **Section ??**, we prove the decidability of quantifier-free matching logic, again via the small-model property, suing a proof similar to that in [Niwiński and Walukiewicz 1996].
- In **Section 6**, we present a translation from matching logic to $\mathrm{FO}_{\mathrm{lfp}}$, and show that this translation preserves satisfiability even in the presence of axioms.
- In **Section 7**, we present guarded matching logic and prove that the image of its translation is in loosely fixedpoint logic.
- In **Section 8**, we show a few theories that fall into this fragment, including modal μ-calculus, finite- and infinite-trace linear temporal logic, CTL, and propositional dynamic logic. We also describe how one my define many-sorted and order-sorted logics within this fragment.

4 Matching Logic Preliminaries

Matching logic was first proposed in [Roşu 2017b] as a unifying logic for specifying and reasoning about programming languages. Matching logic formulae are called *patterns* and have a "pattern matching" semantics, in the sense that each pattern represents the set of elements that "match" it. For example, $\mathsf{cons}(42, x)$ matches lists whose first element is 42, while $\mathsf{prime} \wedge \mathsf{even}$ matches the natural 2 (assuming axiomatizations for cons, prime, and even). Patterns are built using four components: structure for building terms, logic for the usual logical connectives, quantification for first-order quantification, and fixedpoint for building least and greatest fixedpoints.

4.1 Matching Logic Syntax

An important feature of matching logic is that it makes no distinction between terms and formula. This flexibility makes many important concepts easily definable in matching logic, and allows for awkwardness free encodings of various abstractions and logics possible. For example, unification may be characterized by conjuncting two pattern built from constructors.

For a set, EVar of *element variables* ($x, y, z, \ldots$), and SVar of *set variables* ($X, Y, Z, \ldots$), we define the syntax of matching logic below.

Definition 4.1 (Matching logic signatures)**.** A matching logic *signature*, Σ is a set of symbols with an associated arity. Symbols with an arity of zero are called *constants*.

Definition 4.2 (Patterns)**.** Given a signature Σ, a countable set of element variables EVar and of set variables SVar, a matching logic *pattern* is built recursively using the following grammar:

$$\varphi := \underbrace{\sigma(\varphi_1, \ldots, \varphi_n)}_{\text{structure}} \mid \underbrace{\varphi_1 \wedge \varphi_2 \mid \neg\varphi}_{\text{logic}} \mid \underbrace{x \mid \exists x.\, \varphi}_{\text{quantification}} \mid \underbrace{X \mid \mu X.\, \varphi}_{\text{fixedpoint}}$$

where $x \in \mathrm{EVar}$, $X \in \mathrm{SVar}$ and $\sigma \in \Sigma$ has arity n, and X occurs only positively in $\mu X.\, \varphi$. That is, X may only occur under an even number of negations in φ.

We assume the standard notions for free variables, α-equivalence, and capture-free substitution $\varphi[\psi/x]$ and allow the usual syntactic sugar:

$$\top \equiv \exists x.\, x \qquad \varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2) \qquad \forall x.\, \varphi \equiv \neg\exists x.\, \varphi$$
$$\bot \equiv \neg\top \qquad \varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2 \qquad \nu X.\, \varphi \equiv \neg\mu X.\, \neg\varphi[\neg X/X]$$

$\sigma(\varphi_1, \ldots, \varphi_n)$ are called applications. We will use σ instead of $\sigma()$ for constants.

4.2 Semantics of Matching Logic

Unlike in FOL, matching logic patterns are interpreted as a set of elements in a model rather than a single element. Intuitively, the interpretation is the set of elements that match a pattern. For example, the constant even might have as interpretation the set of all odd naturals, while greaterThan(3) may be interpreted as all integers greater than 3. Function symbols may be considered a special case of this, where the interpretation is a singleton set. Logical constructs are thought of as set operations over matched elements – for example, $\varphi \wedge \psi$ is interpreted as the intersection of elements matched by φ and ψ, while $\neg\varphi$ matches all elements *not* matched by φ. An existential $\exists x.\, \varphi(x)$ is interpreted as the union of all patterns matching $\varphi(x)$ for all valuations of x. $\mu X.\, \varphi(X)$ matches the *least* set X such that X and $\varphi(X)$ match the same elements. An important point to note here is that element variables have as denotation exactly a single element, whereas set variables may be interpreted as any subset of the carrier set.

Definition 4.3 (Σ-models)**.** Given a signature Σ, a Σ-model is a tuple $(\mathbb{M}, \{\sigma_M\}_{\sigma\in\Sigma})$ where $\mathbb{M}$ is a set of elements called the carrier set, and $\sigma_M : M^n \to \mathcal{P}(M)$ is the interpretation of the symbol σ with arity n into the powerset of M.

We use M to denote both the model M, and it's carrier set, $\mathbb{M}$. We also tacitly use σ_M to denote the pointwise extension, $\sigma_M : \mathcal{P}(M)^n \to \mathcal{P}(M)$, defined as $\sigma_M(A_1, \ldots, A_n) \mapsto \bigcup_{a_i\in A_i} \sigma_M(a_1, \ldots, a_n)$ for all sets $A_i \subseteq M$.

Definition 4.4 (Semantics of matching logic)**.** Let $\rho : \mathsf{EVar} \cup \mathsf{SVar} \to \mathcal{P}(M)$ be a function such that $\rho(x)$ is a singleton set when $x \in \mathsf{EVar}$, called an evaluation. Then, the denotation of a pattern φ, written $|\varphi|_{M,\rho}$ is defined inductively by:

$$
\begin{aligned}
|\sigma(\varphi_1, \ldots, \varphi_n)|_\rho &= \sigma_M(|\varphi_1|, \ldots, |\varphi_n|) \text{ for } \sigma \text{ of arity } n \\
|\varphi_1 \wedge \varphi_2|_\rho &= |\varphi_1|_\rho \cap |\varphi_2|_\rho \\
|\neg\varphi|_\rho &= M \setminus |\varphi|_\rho \\
|x|_\rho &= \rho(x) \text{ for } x \in \mathsf{EVar} \\
|\exists x.\, \varphi|_\rho &= \bigcup_{a\in M} |\varphi|_{\rho[a/x]} \\
|X|_\rho &= \rho(X) \text{ for } X \in \mathsf{SVar} \\
|\mu X.\, \varphi|_\rho &= \mathsf{LFP}(\mathcal{F}) \text{ where } \mathcal{F}(A) = |\varphi|_{\rho[A/X]} \text{ for } A \subseteq M
\end{aligned}
$$

As seen, σ is interpreted as a relation. Its interpretation σ_M is not a function in the standard FOL sense. We say that σ_M is *functional*, if:

$$
\|\sigma_M(a_1, \ldots, a_n)\| = 1 \quad \text{for all } a_1 \in M_{s_1}, \ldots, a_n \in M_{s_n} \qquad \text{(functional-symbol)}
$$

4.3 Satisfiability and Validity

In this subsection, formally define satisfiability and validity in matching logic[1]. Because of the powerset interpretation of patterns, the notions of satisfiability and validity differ subtly from those in FOL. Because the interpretations of FOL sentences are two-valued – they must be true or false – the notions of satisfiability and validity in a model coincide. Matching logic patterns evaluate to a subset of the carrier set. We say a pattern is satisfiable in a model when its evaluation is non-empty, and that it is valid when its evaluation is the entire carrier set. For example, the model $\mathbb{N}$ with the usual interpretations, satisfies both even and $\neg$even (i.e. the set of odd naturals) but neither are valid.

Definition 4.5 (Satisfiability in a model)**.** We say a Σ-model M *satisfies* a Σ-pattern iff there is some evaluation ρ and an element m such that $m \in |\varphi|_{M,\rho}$. A Σ-pattern φ is *satisfiable* iff there is a model M that satisfies φ.

Definition 4.6 (Validity in a model)**.** We say a Σ-pattern is *valid* in a Σ-model M iff for all evaluations ρ, $|\varphi|_{M,\rho} = M$.

Analogously to FOL, we may define theories in matching logic. Essentially, a theory is a set of patterns, called axioms, that are valid in a model. A pattern is satisfiable modulo a theory if it is satisfiable in some model where all axioms are valid.

Definition 4.7 (Satisfiability modulo theories)**.** Let Γ be a set of Σ-patterns called *axioms*. We say φ is satisfiable modulo theory Γ if there is a model M such that each γ in Γ is valid and M satisfies φ.

4.4 Fragments and Meta-Properties

In general, matching logic's power means that the logic as a whole does not have several desirable properties. For example, because it subsumes first-order logic, the satisfiability problem must be undecidable. Further, because we can precisely pin down the standard model of the natural numbers using the fixedpoint operator, by Gödel's incompleteness theorem, it must also be incomplete. When studying such properties in the context of matching logic, we must thus restrict ourselves to subsets of matching logic. In this section, we shall formally define what each of these properties mean within a subsets, or "fragment", of matching logic.

Definition 4.8 (Fragments of matching logic)**.** A *fragment of matching logic* is a pair $(\mathcal{P}, \mathcal{T})$ where $\mathcal{P}$ is a set of patterns and $\mathcal{P}$ is a set of theories. We say a pattern P is in a fragment if $P \in \mathcal{P}$, and a theory Γ is in a fragment if $\Gamma \in \mathcal{T}$

[1]Note that our definitions differ from [Roşu 2017b] where only validity in a model is defined (but referred to as satisfiability). We avoid using the $\models$ notation to avoid confusion between the two.

Fragments may be defined with any number of criteria, including the restrictions on the use of quantifiers and fixedpoints, number and arity of symbols, the number of axioms, quantifier alternation and so on.

We will now define the properties of fragments of matching logic that we will study in this document.

Definition 4.9 (Decidable fragments)**.** A fragment of matching logic, $(\mathcal{P}, \mathcal{T})$, is *decidable* if deterimining the satisfiability of any pattern $P \in \mathcal{P}$ in any theory $\Gamma \in \mathcal{T}$ is decidable.

Notice that if the $\mathcal{P}$ is closed under negation, then the validity problem for a decidable fragment is also decidable.

For proving the decidability of some fragments in this paper, we rely on a more specific property called the small-model property. This property says that every Γ-satisfiable pattern in a fragment has a model bound by a computable function on the size of the pattern. Formally,

Definition 4.10 (Small-model property)**.** A fragment of matching logic, $(\mathcal{P}, \mathcal{T})$, has the small-model property iff for every pattern $P \in \mathcal{P}$ in every theory $\Gamma \in \mathcal{T}$ if P is Γ-satisfiable then, there is some model $M \models \varphi$ whose size is bound by a computable function f on the size of φ – $\|M\| \leq f(\|\varphi\|)$.

The small-model property implies that a fragment is decidable since one may simply enumerate all models and evaluations for models up to a particular size to prove its decidablity. The small-model property is a stonger version of another interesting property, called the finite-model property:

Definition 4.11 (Finite-model property)**.** A fragment of matching logic, $(\mathcal{P}, \mathcal{T})$, has the finite-model property iff for every pattern $P \in \mathcal{P}$ in every theory $\Gamma \in \mathcal{T}$ if P is Γ-satisfiable then, there is some model $M \models \varphi$ with finite size.

The finite-model property and decidablity are independent in the sense that a fragment may have the finite model property and yet be undecidable, or be decidable despite being infinite.

4.5 The Status-Quo

We will now define some important fragments and enumerate the know results about their meta-properties.

Definition 4.12 (The modal fragment)**.** The *modal fragment* of matching logic has
$\mathcal{P}$ = { patterns built from structure and logic }, and $\mathcal{T} = \{\{\}\}$.

That is, the modal fragment of matching logic only allows quantifier- and fixedpoint-free patterns and the empty theory. This fragment may be regarded as a polyadic multi-arity variant of modal logic. In Section **??**, we show that this fragment has the small-model property (and therefore is also decidable and has the finite-model property).

The quantifier free fragment is less restrictive, allowing fixedpoints in patterns as well:

Definition 4.13 (The quantifier-free fragment)**.** The *quantifier-free fragment* of matching logic has
$\mathcal{P}$ = { patterns built from structure , logic and fixedpoint }, and $\mathcal{T} = \{\{\}\}$.

This fragment also exhibits the small-model property as proved in Section **??**.

We shall only define the next fragment, called guarded matching logic, informally here. We shall describe it in more depth in Section 7. This fragment allows both quantification and fixedpoints. However, quantifiers must be of the form:

$$\forall \bar{x}.\, \alpha(\bar{x}, \bar{y}) \implies \varphi(\bar{x}, \bar{y})$$
$$\exists \bar{x}.\, \alpha(\bar{x}, \bar{y}) \wedge \varphi(\bar{x}, \bar{y})$$

where α is a conjunction of applications and every pair of free variables in φ are arguments of some application in α. This fragment possesses neither the small-model property nor the less strict finite-model property and yet is decidable.

Finally, for the sake of completeness, we also define fixedpoint-free matching logic, and full matching logic, that includes all matching logic patterns and theories. Both these fragments subsume first-order logic and so are neither decidabile, nor have the small- or finite-model properties.

Let us consider, in addition, variants of these fragments that allow different cardinalites of axioms. For a fragment $\mathcal{F}$ that only allows empty theories, we define the fragment $\mathcal{F}_{\text{fin}}$(resp. $\mathcal{F}_{\text{inf}}$) to mean the fragment $(\mathcal{P}, \mathcal{T})$ with $\mathcal{P}$ the same as in $\mathcal{F}$ and $\mathcal{T}$ the set of theories with axioms in $\mathcal{P}$ and finite (resp. recursively enumerable) axioms. For a fragment $\mathcal{F}$ that allows axioms, we define $\mathcal{F}_{\emptyset}$ to only allow empty theories. Similarly, $\mathcal{F}_{\text{inf}}$and $\mathcal{F}_{\text{fin}}$place or loosen the restrictions on the cardinality of the axioms.

In the example below, we show that even the most basic fragment with infinite axioms do not possess any of the properties we consider.

Example 4.14. Consider a signature that contains one sort, one constant symbol z, and two unary symbol s and f. We write $s^0(z)$ to mean z and $s^{n+1}(z)$ to mean $s(s^n(z))$ for $n \geq 1$. Let $\Gamma = \{f(s^n(z)) \mid n \in \mathbb{N}\} \cup \{\neg(s^n(z) \wedge s^m(z)) \mid m, n \in \mathbb{N}, m \neq n\}$ be an infinite theory. Then there exists a model M_0 such that $M_0 \vDash \Gamma$ and for any $M \vDash \Gamma$, we have that M is infinite.

Proof. We first prove that M is infinite for any $M \vDash \Gamma$. Let $z_M \subseteq M$ and $s_M, f_M \colon M \to \mathcal{P}M$ be the interpretations of z, s, and f in M. Since $M \vDash f(s^n(z))$ for every $n \in \mathbb{N}$, and $f(s^n(z))$ is a sentence, we have $|f(s^n(z))| = M$. This implies that $|s^n(z)| \neq \emptyset$. Because $M \vDash \neg(s^m(z) \wedge s^n(z))$ for every m, n with $m \neq n$, we have $|\neg(s^m(z) \wedge s^n(z))| = M \setminus (|s^m(z)| \cap |s^n(z)|) = \text{M}$, which implies that $|s^m(z)| \cap |s^n(z)| = \emptyset$ for every m, n with $m \neq n$. Therefore, $\left|s^0(z)\right|, \left|s^1(z)\right|, \left|s^2(z)\right|, \ldots$ is a sequence of nonempty, pairwisely distinct subsets of M. And thus, M is infinite.

Now, we construct a model M_0 such that $M_0 \vDash \Gamma$. Let $\mathbb{N}$ be domain of M_0. Let $z_{M_0} = \{0\}$, $s_{M_0}(n) = \{n + 1\}$, and

$f_{M_0}(n) = \mathbb{N}$ for $n \in \mathbb{N}$. By mathematical induction, we can prove that $|s^m(z)| = \{m\}$ for all $m \in \mathbb{N}$. By Definition 4.4, we conclude that $M_0 \vDash \Gamma$. □

We summarize the meta properties of these fragments in Table 1.

4.6 A note about variants of matching logic

In its original formulation, matching logic had a many-sorted flavor where each symbol and pattern had a fixed sort. While it is convenient to define models that are also many-sorted, the authors of [Chen and Roşu 2019] pointed out that the many-sorted setting actually becomes an obstacle when it comes to more complex sort structures. Therefore, they proposed a much simpler, unsorted variant of matching logic called applicative matching logic (AML), where the many-sorted infrastructure is dropped and sorts are instead defined axiomatically. This also treated multi-arity applications, as syntactic sugar for nested applications. In this work, to maximize the expressivity of the fragment defined here while still avoiding the complexity of multiple sorts, we use a version of matching logic that sits between the two, allowing multi-arity applications, but without sorts. When we need to be explicit about this distinction, we will refer to this as *polyadic matching logic*. In Section 8, we show that the results presented here also apply to the many-sorted variant, and to AML as well. For the rest of this document unless explicitly mentioned, we will use pattern, model, etc, to refer to those concepts in polyadic matching logic although the same terms may be used in other variants of matching logic.

5 First-order logic with fixedpoints ($\mathrm{FO}_{\mathrm{lfp}}$) preliminaries

$\mathrm{FO}_{\mathrm{lfp}}$, or fixedpoint logic[Chandra and Harel 1982; Gurevich 1984], extends first-order predicate logic (i.e. first-order logic without function symbols) with notation for expressing fixedpoints. Fixedpoints allow expressing reachability problems and properties about algebraic and co-algebraic data types which are important to program verification. Since most readers will be familiar with first-order predicate logic, we will only focus on parts that are different.

Definition 5.1 (First-order signature)**.** A *signature* Σ is a finite set of relation symbols $P, Q, R, \ldots$ and constant symbols $c, d, \ldots$. Each relation symbol is equipt with a natural number called its arity.

Definition 5.2 ($\mathrm{FO}_{\mathrm{lfp}}$ syntax)**.** Let V be a countable set of *(first-order) variables*, and W be a countable set of *relation variable* each with an associated arity. Then, for a (first-order) signature Σ, the syntax of $\mathrm{FO}_{\mathrm{lfp}}$ is defined as:

$t :=$
- x where x is a variable
- $\mid c$ where c is a constant in Σ

$\varphi :=$ (first-order predicate logic syntax)
- $\mid \left[\mathsf{LFP}_{W,\bar{x}}.\, \varphi\right](\bar{t})$ where W is an n-ary relation variable, and $\bar{x}$ and $\bar{t}$ have length n, and W occurs only positively in φ. Further, $\bar{x}$ are the only free first-order variables in φ.
- $\mid W(t_1, t_2, \ldots, t_n)$ where W is an n-ary relation variable.

In $\left[\mathsf{LFP}_{W,\bar{x}}.\, \varphi\right](\bar{t})$, the relation variable W and each $x \in \bar{x}$ are bound, while variables in $\bar{t}$ are considered free. Relational variables may only occur bound in a fixedpoint construct.

Definition 5.3 ($\mathrm{FO}_{\mathrm{lfp}}$models)**.** For a signature Σ, a model $\mathcal{A}$, is a non-empty set A with a relation $R_A \subseteq A^n$ for every n-ary relation symbol R in Σ and an element c_A for every constant c in Σ.

Definition 5.4 (Valuation)**.** A valuation in a structure $\mathcal{A}$ is a function α from variables to elements in A, and from n-ary relation variables to n-ary relations. Valuations are extended to terms by setting $\alpha(c) = c_A$.

Definition 5.5 ($\mathrm{FO}_{\mathrm{lfp}}$ semantics)**.** An valuation α satisfies the formula φ in structure $\mathcal{A}$ (written $\mathcal{A}, \alpha \models \varphi$), iff in addition to the usual first-order predicate logic semantic rules:

$$\mathcal{A}, \alpha \models W(t_1, \ldots, t_n) \quad \text{iff } \alpha(W)(\alpha(t_1), \ldots, \alpha(t_n))$$

$$\mathcal{A}, \alpha \models \left[\mathsf{LFP}_{W,\bar{x}}.\, \varphi\right] \quad \text{iff } (\alpha(t_1), \ldots, \alpha(t_n)) \in \mathsf{LFP}(\varphi^{\mathcal{A},\alpha})$$

where $\varphi^{\mathcal{A},\alpha}$ is defined below.

Let α be an valuation providing interpretations for all free relation variables in φ except W. Then φ defines an operator:

$$\varphi^{\mathcal{A},\alpha} : A^n \to A^n$$

$$\varphi^{\mathcal{A},\alpha}(S) \mapsto \{a \in A^n : \mathcal{A}, [W \mapsto S, x \mapsto a] \models \psi\}$$

Since W occurs only positively, this operator is monotone and has a least fixedpoint.

6 Translation between matching logic and $\mathrm{FO}_{\mathrm{lfp}}$

In this section, we define a satisfiability-preserving translation between matching logic and $\mathrm{FO}_{\mathrm{lfp}}$. This translation is similar to the translation to predicate logic presented in [Roşu 2017b], but includes the fixedpoint operator and focuses on satisfiability rather than validity. Importantly, our translation works with axioms as well, allowing us to translate matching logic theories into $\mathrm{FO}_{\mathrm{lfp}}$ preserving satisfiability.

While our translation is quite general, there are two restriction we have on patterns:

Property \ Fragment	Modal	Quantifier-free	Guarded	Fixedpoint-free	Full
Empty theories					
Small-model	✓[Sec.??]	✓[Sec.??]	✗	✗	✗
Finite-model	✓	✓	✗[Ex.7.5]	✗	✗
Decidability	✓	✓	✓	✗	✗
Finite theories					
Small-model	?	?	✗	✗	✗
Finite-model	?	?	✗	✗	✗
Decidability	✓[Sec.7]	✓†[Sec.7]	✓[Sec.7]	✗	✗
Recursively enumerable theories					
Small-model	✗	✗	✗	✗	✗
Finite-model	✗[Ex. 4.14]	✗	✗	✗	✗
Decidability	✗[Urquhart 1981]	✗	✗	✗	✗

Table 1. *The status quo:* Fragments of matching logic and their meta-prorties.
† This result has only been proved when there are no free set variables in axioms.

Definition 6.1 (Translatable patterns)**.** Translatable patterns are matching logic patterns where:

1. No free element variables are used in fixedpoint operators
2. All set variables are bound by a fixedpoint operator.

Requirement (1) is due to a similar restriction in $\mathrm{FO}_{\mathrm{lfp}}$ disallowing free first-order variables in fixedpoints. Requirement (2) is needed because free set variables in axioms are implicitly universally quantified. Translating these to $\mathrm{FO}_{\mathrm{lfp}}$ would require second order quantification. This is not an issue when set variables occur in the main pattern whose satisfiability is being checked. This is because we may replace those set variables with nullary symbols, similar to Skolemization.

For a matching logic signature, Σ, we define a $\mathrm{FO}_{\mathrm{lfp}}$ signature, $\Sigma_{\mathrm{FO}_{\mathrm{lfp}}}$ with relation symbols R_σ of arity $n+1$ for every symbol $\sigma \in \Sigma$ of arity n.

For translatable patterns, we define two functions:

$$\begin{aligned} \mathsf{sat?}(\_) &: \mathsf{Pattern} \to \mathsf{Formula} \\ \mathsf{sat?}(\varphi) &\mapsto \exists x, \mathsf{free}(\varphi).\, \mathsf{matches?}(x, \varphi) \\ &\qquad \text{where } x \text{ is fresh.} \end{aligned}$$

and:

$$\begin{aligned} \mathsf{matches?}(\_,\_) &: \mathsf{Var} \times \mathsf{Pattern} \to \mathsf{Formula} \\ \mathsf{matches?}(x, \sigma(\varphi_1, \ldots, \varphi_n)) &\mapsto \exists x_1, \ldots, x_n.\, R_\sigma(x, x_1, \ldots x_n) \\ &\qquad \wedge \bigwedge_{i \le n} \mathsf{matches?}(x_i, \varphi_i) \\ &\qquad \text{where each } x_i \text{ is fresh.} \\ \mathsf{matches?}(x, \varphi_1 \wedge \varphi_2) &\mapsto \mathsf{matches?}(x, \varphi_1) \\ &\qquad \wedge \mathsf{matches?}(x, \varphi_2) \\ \mathsf{matches?}(x, \neg\varphi) &\mapsto \neg\mathsf{matches?}(x, \varphi) \\ \mathsf{matches?}(x, y) &\mapsto x = y \\ \mathsf{matches?}(x, \exists y.\, \varphi) &\mapsto \exists y.\, \mathsf{matches?}(x, \varphi) \\ \mathsf{matches?}(x, Y) &\mapsto Y(x) \\ \mathsf{matches?}(x, \mu Y.\, \varphi) &\mapsto \left[\mathsf{LFP}_{Y,y}.\, \mathsf{matches?}(y, \varphi)\right](x) \end{aligned}$$

These functions are well defined and preserve satisfiability:

Proposition 6.2. *The above translation is well-defined. (i.e. the translated formulae are well-formed FO_{lfp} formulae)*

Proof. This can be proved by induction on the the structure of translatable patterns. The only interesting case for μ. First we show that in fixedpoint formulae there can be no free element variables besides those bound by the fixedpoint construct. A fixedpoint pattern is translated as $\mathsf{matches?}(x, \mu Y.\, \varphi) \mapsto \left[\mathsf{LFP}_{Y,y}.\, \mathsf{matches?}(y, \varphi)\right](x)$. Since, in Requirement (2), we restrict translatable patterns to disallow free elemental variables in fixedpoints, only y is free in $\mathsf{matches?}(y, \varphi)$. The syntactic requirements of matching logic require that bound set variables occur only positively in fixedpoint construct, and the fact that negations are preserved by the translation and no new negations are introduced means that relational variables occur only positively in the LFP operator. Finally,

every relational variable must be bound by an LFP operator. This must hold because of Requirement (2). □

Proposition 6.3. *For the empty theory* $\text{sat?}(\varphi)$ *is satisfiable in* FO_{lfp} *iff* φ *is satisfiable in matching logic.*

Proof Sketch. (See Appendix. A for the full proof.) This is proved by defining a matching logic model for any FO_{lfp} model, and vice-versa, and then showing that if a pattern/-formula is satisfiable in one model then it is satisfiable in the other. □

Let us give a few example translations. The pattern $\text{cons}(a, \text{cons}(b, c))$ which represents a cons list of length two is translated as follows:

$$\exists x, a, b, c.\, \exists x_1, x_2.\, R_{\text{cons}}(x, x_1, x_2) \wedge x_1 = a \wedge \exists x_{21}, x_{22}.\, R_{\text{cons}}(x_2, x_{21}, x_{22}) \wedge x_{21} = b \wedge x_{22} = c$$

The pattern $\mu X.zero \vee s(X)$, representing the set of naturals is translated as:

$$\exists m.\, \left[\text{LFP}_{X,x}.\, R_{\text{zero}}(x) \vee \exists x_1.\, R_{\text{s}}(x, x_1) \wedge X(x_1)\right](m)$$

Note that cons, zero and s need to be axiomatized as constructors for the standard interpretation to hold. The complexity of the resultant formulae compared to the relative simplicity of the original matching logic pattern highlights the advantages of matching logic.

6.1 Translation of theories

This translation may be extended to handle satisfiability modulo theories. For a finite theory with translatable axioms Γ and a translatable pattern φ we define:

$$\text{sat}_\Gamma?(\varphi) = \text{sat?}(\varphi) \wedge \bigwedge_{\gamma \in \Gamma} \neg\text{sat?}(\neg\gamma)$$

Theorem 6.4. $\text{sat}_\Gamma?(\varphi)$ *is satisfiable in* FO_{lfp} *iff* φ *is satisfiable in matching logic.*

Proof. A pattern φ is satisfiable in a theory Γ if there is a model M such that for every $\gamma \in \Gamma$ and valuation ρ', $|\gamma|_{\rho'} = M$ and for some ρ, $|\varphi|_\rho \neq \emptyset$. Translatable patterns do not allow free set variables. Free element variables in axioms may be assumed bound using universal quantifiers. So, we may assume all axioms $\gamma \in \Gamma$ and φ are closed. For closed patterns, we only need to consider the satisfiability or validity under the empty evaluation instead of all possible evaluations. A pattern φ is valid iff $|\varphi| = M$. Thus it is valid iff $|\neg\varphi| = M \setminus |\varphi| = \emptyset$, i.e. φ is valid iff its negation is not satisfiable. Thus we have $\text{sat}_\Gamma?(\varphi)$ iff φ is satisfiable in Γ. □

Now that we have a translation from matching logic to fixedpoint logic, use existing results in that logic to derive decidability results for matching logic.

7 Guarded matching logic

In this section, we present the main contribution of our paper – we describe a fragment of matching logic that is decidable. This is proven via a linear translation to the loosely guarded fragment of FO_{lfp}, a decidable fragment. We also allow translation of axioms in theories if they are in the described fragment. First, we give an overview of the loosely guarded FO_{lfp} as defined in [Gradel and Walukiewicz 1999].

7.1 Guarded fixedpoint logic

Guarded first-order logic is a decidable fragment of FOL, developed by [Andréka et al. 1998]. The key insight was that the undecidability of first order logic stemmed from the unbounded nature of quantification, and that by restricting the elements we may quantify over, we may define a decidable fragment. In this fragment of FOL, quantification is restricted to the form $\exists \bar{y}.\, \alpha(\bar{x}, \bar{y}) \wedge \psi(\bar{x}, \bar{y})$, where α, called the guard, is an atom that mentions all free and quantified variables of ψ. In [Hodkinson 2002], this restriction was relaxed to allow conjunctions of atoms, and extended by [Gradel and Walukiewicz 1999] to allow fixedpoints in ψ (but not in the guard). We will use this fragment as the basis of our decidability results.

Definition 7.1 (Loosely guarded FO_{lfp})**.** Loosely guarded FO_{lfp} is the fragment of FO_{lfp} defined inductively as follows:

1. Every relational atom and equality is in loosely guarded FO_{lfp},
2. loosely guarded FO_{lfp} is closed under logical connectives $\neg, \wedge, \vee, \rightarrow$ and $\leftrightarrow$, and
3. If ψ is in loosely guarded FO_{lfp}, then $\left[\text{LFP}_{W,\bar{x}}.\, \psi\right](\bar{t})$ where $\text{free}(\psi) = \bar{x}$
4. If ψ is in loosely guarded FO_{lfp}, and $\alpha \equiv \alpha_1 \wedge \ldots \wedge \alpha_n$ is a conjunction of atoms and equalities, then

$$\exists \bar{y}.\, \alpha(\bar{x}, \bar{y}) \wedge \psi(\bar{x}, \bar{y})$$
$$\forall \bar{y}.\, \alpha(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{y})$$

belongs to loosely guarded FO_{lfp}, provided that $\text{free}(\psi) \subseteq \text{free}(\alpha) = \bar{x} \cup \bar{y}$ and for every quantified variable $y \in \bar{y}$ and every variable $x \in \bar{x} \cup \bar{y}$, there is at least one atom α_j that contains both y and z.

Let us recall the decidability result for loosely guarded FO_{lfp} as proved in [Gradel and Walukiewicz 1999]:

Theorem 7.2. *The satisfiability problem for guarded fixedpoint sentences is 2EXPTIME-complete.*

When restricted to a fixed finite vocabulary, formulae have bounded width – i.e. the number of free variables in sub-formulae are bounded. This allows a tighter bound on the complexity:

Theorem 7.3. *The satisfiability problem for guarded fixedpoint sentences of bounded width is EXPTIME-complete.*

In the next subsection we will define a syntactic fragment of matching logic that translates into loosely guarded fixed-point logic.

7.2 Guarded matching logic

We consider a syntactic fragment of matching logic that translates to loosely guarded $\mathrm{FO}_{\mathrm{lfp}}$, and so is decidable. This fragment allows quantifiers, although restricted similarly to loosely guarded $\mathrm{FO}_{\mathrm{lfp}}$. Applications are also restricted as described below. In the interest of conciseness we call it guarded matching logic, instead of loosely guarded matching logic.

Definition 7.4 (Guarded pattern)**.** A guarded pattern is a translatable pattern constructed recursively using:

1. element variables bound by point (5) below.
2. logic and fixedpoint
3. applications whose arguments are guarded and employ no free variables,
4. applications of the form $\sigma(\bar{\varphi})$, where each $\varphi \in \bar{\varphi}$ is of the form $x \wedge \psi$ and x is an element variable bound by point (5) and ψ is guarded.
5. patterns of the form $\exists \bar{x}.\, \alpha(\bar{x}, \bar{y}) \wedge \varphi(\bar{x}, \bar{y})$ where:
   - φ is a guarded matching logic pattern,
   - α is a conjunction of patterns of the form $\sigma(\bar{x})$, where $\bar{x}$ are all element variables,
   - $\mathsf{free}(\varphi) \subseteq \mathsf{free}(\alpha) = \bar{x} \cup \bar{y}$,
   - for each quantified variable $x \in \bar{x}$ and free variable $v \in \bar{x} \cup \bar{y}$ occurs together as arguments to an application in the guard α, and

Let us give a few examples of guarded pattern. First, notice that all quantifier-free patterns are guarded. The pattern $f(\top) \equiv f(\exists x.\, x)$ is guarded, since the inner argument has no free variables.

$f(g(y), h())$ is *not* guarded. It translates to $\exists x_1, x_2.\, R_f(x, x_1, x_2) \wedge R_g(x_1, y) \wedge R_h(x_2)$. Notice that the x_2 and y do not occur simultaneously in any atom of the guard. Such patterns are the reason for the strictness of the criteria in point (4). The pattern $f(g(y), z)$, while not a guarded *pattern*, translates to a guarded fixedpoint formula: $\exists x_1.\, R_f(x, x_1, z) \wedge R_g(x_1, y)$, so our criteria aren't quite exact.

Guarded matching logic does not posess the small-model property, or even the finite model property, as demonstrated by the following example.

Example 7.5.

$$\mathrm{zero} \vee \mathrm{succ}(\top)$$
$$\mu N.\, \mathrm{zero} \vee \mathrm{succ}(N)$$
$$\neg \mathrm{zero} \wedge \mathrm{succ}(\top)$$
$$\forall x, y.\, \mathrm{succ}(x) \wedge \mathrm{succ}(y) \rightarrow \mathrm{succ}(x \wedge y)$$

Before we can proof that guarded patterns translate to loosely guarded formulae, we must introduce a transformation that removes equalities from guards. Loosely guarded $\mathrm{FO}_{\mathrm{lfp}}$ formulae do not allow equalities in guards. However, even the most basic matching logic patterns have a translation that uses such equalities (because an element variable z is translated to $x = z$). So, we must define another transformation that removes them when possible.

Definition 7.6 (Normalization)**.** We define a linear translation from $\mathrm{FO}_{\mathrm{lfp}}$ formulae to $\mathrm{FO}_{\mathrm{lfp}}$ formulae as:

$$\begin{aligned}&\mathrm{normalize}(\exists x, \bar{x}.\, (x = y \wedge \alpha) \wedge \varphi)\\ &\quad = \mathrm{normalize}(\exists \bar{x}.\, \alpha[y/x] \wedge \varphi[y/x])\\ &\mathrm{normalize}(\forall x, \bar{x}.\, (x = y \wedge \alpha) \rightarrow \varphi)\\ &\quad = \mathrm{normalize}(\forall \bar{x}.\, \alpha[y/x] \rightarrow \varphi[y/x])\end{aligned}$$

The function recurses on the structure of the pattern in the usual sense for all other cases. Since the number of equalities in quantifiers is finite, this is well-defined.

Let us now prove the main result:

Theorem 7.7. *The satisfiability problem for guarded matching logic is decidable.*

via the following proposition:

Proposition 7.8. *After normalization, guarded matching logic patterns translate to loosely guarded FO_{lfp} formulae.*

Proof. We must consider all patterns that may introduce existentials either implicitly through applications, or through explicit quantification. First, let us consider a pattern $\sigma(\varphi_1, \ldots, \varphi_n)$ that meets the criteria under point 3 – i.e. an application whose arguments do not contain free variables. We have:

$$\begin{aligned}\mathsf{matches?}(x, \sigma(\varphi_1, \ldots, \varphi_n)) \;\mapsto\; &\exists x_1, \ldots, x_n.\, R_\sigma(x, x_1, \ldots x_n)\\ &\wedge \textstyle\bigwedge_{i \le n} \mathsf{matches?}(x_i, \varphi_i)\end{aligned}$$

The atomic term $R_\sigma(x, x_1, \ldots x_n)$ guards all free variables in this pattern, and so it is in loosely guarded $\mathrm{FO}_{\mathrm{lfp}}$.

Next, let us consider an application that use element variables under the criteria for point 4. We have:

$$\begin{aligned}&\mathrm{normalize}(\mathsf{matches?}(m, \sigma(x_1 \wedge \varphi_1, \ldots, x_n \wedge \varphi_n)))\\ \equiv\ &\mathrm{normalize}(\exists \bar{y}.\, R_\sigma(m, \bar{y}) \wedge \bigwedge y_i = x_i \wedge \mathsf{matches?}(y_i, \varphi_i))\\ \equiv\ &R_\sigma(m, \bar{x}) \wedge \bigwedge \mathsf{matches?}(x_i, \varphi_i[x_i/y_i])\end{aligned}$$

Applications of this form do not introduce any existentials, and so are guarded by induction.

Finally, let us consider explicit existentials meeting the criteria in point (5). Each introduced existential translates as follows:

$$\begin{aligned}&\mathsf{matches?}(m, \exists \bar{x}.\, \bigwedge \sigma_i(\bar{a}_i) \wedge \varphi(\bar{x}, \bar{y}))\\ \equiv\ &\exists \bar{x}.\, \mathsf{matches?}(m, \bigwedge \sigma_i(\bar{a}_i) \wedge \varphi(\bar{x}, \bar{y}))\\ \equiv\ &\exists \bar{x}.\, \bigwedge \mathsf{matches?}(m, \sigma_i(\bar{a}_i)) \wedge \mathsf{matches?}(m, \varphi(\bar{x}, \bar{y}))\\ \equiv\ &\exists \bar{x}.\, \bigwedge R_{\sigma_i}(m, \bar{a}_i) \wedge \mathsf{matches?}(m, \varphi(\bar{x}, \bar{y}))\end{aligned}$$

Here, $\bigwedge R_{\sigma_i}(m, \bar{a}_i)$ forms the guard. The free and quantified variables meet the criteria for a guard for loosely guarded $\mathrm{FO}_{\mathrm{lfp}}$ because of point (5). □

Now that we have defined a decidable fragment of matching logic, we will show that this fragment is useful by showing a few theories that fall into it.

8 Decidable theories

In this section, we will prove that a few known theories and logics lie in the decidable fragments described above. While all of these theories and logics were previously known to be decidable, and indeed inspired our work, it is in our view practically important and theoretically interesting that each of these proofs reduce to simple syntactic checks, without any cumbersome encodings or translations: they are direct instances of the more general decidability result for matching logic, including their decidability proofs. Previously, proving these results would have involved developing such proofs from scratch. This makes developing of new abstractions and logics much simpler and practical, and motivates using matching logic as a lingua franca and foundation on which other abstractions and logics may be built. The working logician does not need to worry that adding a new construct, or modifying the logic slightly, will require to understand and redo the decidability proof: if the new logic is a syntactic instance of guarded matching logic then it is decidable and a direct proof can be extracted from the guarded matching logic decidability proof.

Since matching logic strives to minimize representational distance when working with an embedded logic, it allows user-defined "notation" to represent syntactic sugar. Notation is a map between sugar, defined using meta-variables (e.g. φ and ψ), and the patterns they represent. For example, the notation $\circ\varphi \equiv \neg\bullet(\neg\varphi)$ defines $\circ$ (one-path next) as sugar for $\bullet$ (all-path next). Importantly, $\circ$ is *not* a symbol in its own right.

The theories and logics defined below will include three parts: a list of symbols defining their signature, a list of notation, and a list of axioms constraining the models.

8.1 Modal μ-calculus

Modal μ-calculus extends modal logic with fixedpoints which in turn extends propositional logic with "actions" or "modalities". Formulae are interpreted in a set of "worlds", each labeled with atomic propositions. For each modality, these worlds are connected through a successor relation. For a set of propositional variables PVar and modalities A, modal-μ calculus formulae are defined by the following grammar:

$$\begin{array}{ll} \varphi := p & \text{where } p \in \mathsf{PVar} \\ \quad \mid \varphi \wedge \varphi \mid \neg\varphi & \\ \quad \mid [a]\varphi \mid \langle a\rangle\varphi & \text{where } a \in A \\ \quad \mid \mu X.\, \varphi \mid \nu X.\, \varphi & \text{where } X \text{ positive in } \varphi \end{array}$$

A propositional variable holds if the current world is labeled with that variable. Logical operators are interpreted as usual. $\langle a\rangle\varphi$ holds if φ holds in *any* a-successor of the current world, while $[a]\varphi$ holds if φ holds in *every* a-successor. μ and ν are interpreted as the least and greatest fixedpoints. For example, $\nu X.\, \varphi \wedge [a]X$ may be interpreted as "φ holds along every a-path".

Modal μ-calculus is embedded in matching logic with a unary symbol for each modality and a nullary symbol for each propositional variable [Chen and Roşu 2019]:

theory MODAL–MU–CALCULUS
  Symbols:
    $a(\_)$ for each $a \in A$
    p for each $p \in \mathsf{PVar}$
  Notation:
    $\langle a\rangle\varphi \equiv a(\varphi)$
    $[a]\varphi \equiv \neg a(\neg\varphi)$
endtheory

No axioms are needed. The decidability of modal μ-calculus may now be proved through a simple one-line proof:

Proposition 8.1. *Modal μ-calculus is decidable.*

Proof. All formulae translate to the quantifier free patterns and since it uses no axioms, this logic is a sub-fragment of quantifier-free and guarded fragments of matching logic. So Modal μ-calculus is decidable. □

8.2 Infinite trace LTL

Linear temporal logic is an important logic in program verification. It takes as models infinite non-branching traces. For a set of propositional variables PVar, LTL formulae are defined by the following grammar:

$$\varphi := p \in \mathsf{PVar} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \circ\varphi \mid \varphi\ U\ \varphi$$

Again, an embedding in matching logic is possible by treating LTL formulae as syntactic sugar for matching logic constructs [Chen and Roşu 2019]. We define a matching logic signature with a single unary symbol $\bullet$, one-path next, and constants for each $a \in \mathsf{PVar}$. The symbol $\bullet$ specifies what may happen on one particular next state, while $\circ$ specifies what may happen on all next states. In infinite trace LTL these two notions are identical. However, for consistency with other matching logic theories that define transition systems, we define one-path next as a symbol, and all path next using notation. All LTL constructs are defined as syntactic sugar:

theory INFINITE–LTL
  Symbols:
    $\bullet(\_)$
    p for each $p \in \mathsf{PVar}$
  Notation:
    (all –path next) $\circ\varphi \equiv \neg\bullet(\neg\varphi)$
    (eventually) $\Diamond\varphi \equiv \mu X.\, \varphi \vee \circ X$

(always) $\Box\varphi \equiv \nu X.\, \varphi \wedge \circ X$
(strong until) $\varphi\ U\ \psi \equiv \mu X.\, \psi \vee (\varphi \wedge \bullet(X))$
Axioms:
(Infinite) $\bullet\top$
(Linear) $\bullet X \rightarrow \circ X$
endtheory

Theorem 33 in [Chen and Roşu 2019] shows that satisfiability and validity in this theory is equivalent to satisfiability in linear LTL. Observe that while every LTL formula corresponds to a pattern that is quantifier-free, only (Infinite) is in the guarded fragment. We may however replace (Linear) with an equivalent guarded axiom:

$$(\mathsf{Linear}') \quad \forall x.\, \bullet x \rightarrow \circ x$$

In (Linear), X ranges over all subsets of the carrier set. Suppose (Linear) is valid in a model. Then (Linear') must also be valid, since x ranges over all singleton sets. Suppose (Linear) is *not* valid in a model. Then there must be some interpretation such that $|\bullet X| \not\subseteq |\circ X|$. There must be some element $e \in |\bullet X|$, $e \notin |\circ X|$. Since the denotation of applications is given by pointwise extension, there is some element x, such that $e \in \bullet_M(x)$. So (Linear') must also not be valid.

Using this theory, we may now prove that infinite trace LTL is decidable:

Proposition 8.2. *Infinite trace LTL is decidable.*

Proof. All formulae translate to the quantifier free patterns, and so are in the guarded fragment. (Linear') only use a single element variable and so the empty guard suffices. (Infinite) is also guarded. The theory lies in guarded matching logic and so is decidable. □

8.3 Finite trace LTL

Finite trace LTL is identical to infinite trace LTL in syntax. It's semantics however, only allows models where all traces are finite. A theory for finite trace LTL is defined in a similar manner to the infinite version in [Chen and Roşu 2019], except that the (Infinite) axiom is replaced with one that forces finite models:

theory FINITE–LTL
Symbols:
$\bullet(\_)$
p for each $p \in \mathsf{PVar}$
Notation:
(all –path next) $\circ\varphi \equiv \neg \bullet (\neg\varphi)$
(eventually) $\Diamond\varphi \equiv \mu X.\, \varphi \vee \circ X$
(always) $\Box\varphi \equiv \nu X.\, \varphi \wedge \circ X$
(strong until) $\varphi\ U\ \psi \equiv \mu X.\, \psi \vee (\varphi \wedge \bullet(X))$
Axioms:
(Finite) $\mu X.\, \circ(X)$
(Linear) $\bullet X \rightarrow \circ X$
endtheory

We may easily prove the decidability of this theory:

Proposition 8.3. *Finite trace LTL is decidable.*

Proof. All formulae translate to the quantifier free patterns, and so are in the guarded fragment. (Finite) is quantifier-free. As before (Linear') may replace (Linear) and is guarded. The theory lies in guarded matching logic and so is decidable. □

Again, we'd like to emphasize that how simple this proof is. The difference between the axioms of finite and infinite trace LTL also highlight the ability to modularly define theories in matching logic, further motivating it as a lingua franca.

8.4 Computational Tree Logic

CTL models are transition systems that are infinite and allow branching (unlike in LTL). The syntax for CTL is as follows:

$$\varphi := p \in \mathsf{PVar} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid AX\varphi \mid EX\varphi \mid \varphi\, AU\, \varphi \mid \varphi\, EU\, \varphi$$

with the following derived constructs:

$$EF\varphi \equiv \mathsf{true}\ EU\ \varphi \qquad EG\varphi \equiv \neg\, AF\, \neg\varphi$$
$$AF\varphi \equiv \mathsf{true}\ AU\ \varphi \qquad AG\varphi \equiv \neg\, EF\, \neg\varphi$$

CTL is defined in matching logic in the following theory:

theory CTL
Symbols:
$\bullet(\_)$
p for each $p \in \mathsf{PVar}$
Notation:
$AX\varphi \equiv \circ\varphi$ $EG\varphi \equiv \neg\, AF\, \neg\varphi$
$EX\varphi \equiv \bullet\varphi$ $AG\varphi \equiv \neg\, EF\, \neg\varphi$
Axioms:
(Infinite) $\bullet\top$
endtheory

Proposition 8.4. *CTL is decidable.*

Proof. All formulae translate to the quantifier free patterns, and so are in the guarded fragment. (Infinite) is guarded as well. The theory lies in guarded matching logic and so is decidable. □

8.5 Propositional dynamic logic

Propositional dynamic logic (PDL) is a logic used for program reasoning. Its syntax is defined by the following grammar:

$$\varphi := p \in \mathsf{PVar} \mid \varphi \rightarrow \varphi \mid \mathsf{false} \mid [\alpha]\varphi$$
$$\alpha := a \in \mathsf{APgm} \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha? \mid \alpha^*$$

This may be embedded in matching logic with the following equivalences:

theory PDL
Symbols:
$\bullet(\_,\_)$
a for each $a \in \mathsf{APgm}$
p for each $p \in \mathsf{PVar}$
Notation:
$\langle\alpha\rangle\varphi \equiv \bullet(\alpha, \varphi)$ $[\alpha]\varphi \equiv \neg\langle\alpha\rangle\neg\varphi$

$$[\alpha;\beta]\varphi \equiv [\alpha][\beta]\varphi \qquad [\alpha \cup \beta]\varphi \equiv [\alpha]\varphi \wedge [\beta]\varphi$$
$$[\psi?]\varphi \equiv (\psi \to \varphi) \qquad [\alpha^*]\varphi \equiv \nu X.\,(\varphi \wedge [\alpha]X)$$

endtheory

No axioms are needed.

Proposition 8.5. *PDL is decidable.*

Proof. All formulae translate to the quantifier free patterns, and since no axioms are used this logic is a sub-fragment of the quantifier-free and guarded fragments. □

8.6 Guarded sorted logics

In this paper, we chose to demonstrate our results on an unsorted variant of matching logic we called polyadic matching logic (PML). However, these results apply equally to applicative matching logic (AML) and to many-sorted matching logic (MSML). AML, although equally expressive, may be considered a syntactic fragment of polyadic matching logic with a single binary symbol of application, and constants. The results presented here translate directly into AML. However, since multi-arity applications in PML translate to nested applications in AML, the guarded fragment of AML is significantly less powerful. We will not show the results for AML in detail because they are identical to those of PML.

MSML annotates each symbol with sorts for each argument and a return sort in addition to an arity. MSML has syntax similar to polyadic matching logic, with some constructs parametric over sorts. For a non-empty set of sorts S, countably infinite set of element and set variables EVar and SVar, the syntax of MSML is defined inductively as follows:

$$\varphi_s := \overbrace{\sigma(\varphi_{s_1},\ldots,\varphi_{s_n})}^{\text{structure}} \mid \overbrace{\varphi_1{:}s \wedge \varphi_2{:}s \mid \neg\varphi}^{\text{logic}}$$
$$\mid \underbrace{x{:}s \mid \exists x{:}s.\,\varphi_s}_{\text{quantification}} \mid \underbrace{X{:}s \mid \mu X{:}s.\,\varphi_s}_{\text{fixedpoint}}$$

where $s \in S$, $x \in$ EVar, $X \in$ SVar, and σ has arity n with argument sorts $s_1,\ldots,s_n$ and return sort s. MSML models have as its sort-indexed universe – each element in the universe has an associated sort. A pattern of a particular sort is interpreted as a set of elements in that sort. The semantics MSML closely parallel polyadic matching logic presented here, and so we will refer the reader to [Chen and Roşu 2019] for a detailed description. Although many sorted matching logic may be captured axiomatically as a guarded theory, we choose to instead give an exposition of various instruments that may be put together to form different varieties of sorted logics, such as many-sorted and order-sorted logics based on [Chen et al. 2021b].

First, we need to set up some important technical instruments for defining equality, set membership and subsets. This is done using the definedness symbol $\lceil\_\rceil$. Intuitively, the definedness symbol evaluates to $\top$ when passed an argument that evaluates to any non-empty set. We may use it to define equality, membership and subset relations. The axiom (definedness) is a guarded pattern.

theory DEFINEDNESS
Symbols:
$\lceil\_\rceil$
Notation:
$\lfloor\varphi\rfloor \equiv \neg\lceil\neg\varphi\rceil$
$\varphi = \psi \equiv \lfloor\varphi \leftrightarrow \psi\rfloor$
$\varphi \neq \psi \equiv \neg(\varphi \neq \psi)$
$\varphi \in \psi \equiv \lceil\varphi \wedge \psi\rceil$
$\varphi \subseteq \psi \equiv \lfloor\varphi \to \psi\rfloor$
Axiom:
(Definedness) $\forall x.\,\lceil x\rceil$
endtheory

Using this, we may now define sorts. For each sort s, we define a constant $[\![s]\!]$, pronounced "inhabitants of s". The intended denotation of this symbol is the set of all elements of the set – its carrier set. We may now define a basic specification for sorts:

theory SORTS
Imports: DEFINEDNESS
Symbols:
$[\![s]\!]$ for each sort s
Notation:
$\neg_s\varphi \equiv [\![s]\!] \wedge \neg\varphi$
$\forall x : s.\,\varphi \equiv \forall x.\,(x \in [\![s]\!]) \to \varphi$
$\exists x : s.\,\varphi \equiv \exists x.\,(x \in [\![s]\!]) \wedge \varphi$
Axiom:
For each sort s:
(Non–empty inhabitants) $[\![s]\!] \neq \bot$
endtheory

We use "Imports: DEFINEDNESS" to indicate that all symbols, notation, and axioms from DEFINEDNESS are included in SORTS. Here $\neg_s\varphi$ is called sorted negation. Intuitively, it matches any element of sort s that does not match φ. Since this notation conjuncts a pattern with a constant, it does not affect its guardedness. Notations $\forall x : s.\,\varphi$ and $\exists x : s.\,\varphi$ are called sorted quantification, where x ranges only over elements of the sort. Similarly, $(s \in [\![s]\!])$ is a guarded pattern, not affecting the guardedness of the pattern. We may simply commute this pattern with the guard to get a guarded quantifier in its canonical form. The axioms (Non-empty inhabitants), equivalent to $\lceil[\![s]\!]\rceil$, specifies that each sort is non empty. These axioms, too, are in the guarded fragment.

Next, we add axioms making each symbol sort appropriately. For a symbol σ that takes arguments $s_1,\ldots,s_n$ and returns sort s we define the axiom:

$$\text{(well-sorted)} \qquad \sigma([\![s_1]\!],\ldots,[\![s_n]\!]) \to [\![s]\!]$$

This quantifier-free axiom is guarded.

In many-sorted logics, we desire that sorts do not intersect – no element is in two sorts simultaneously. For every pair

of sorts s_1 and s_2, we may define the axiom:

$$\text{(distinct sorts)} \qquad \neg(\llbracket s_1 \rrbracket \wedge \llbracket s_2 \rrbracket)$$

Again, this axiom is guarded!

We may decide to instead implement an order-sorted signature, where some sorts are sub-sorted into others:

$$\text{(subsort)} \qquad \llbracket s_1 \rrbracket \subseteq \llbracket s_2 \rrbracket$$

Being quantifier free, this is guarded.

8.7 Developing new logics and theories

The modularity of matching logic allows us to easily develop new logics. For example, we may easily create a finite- or infinite-trace only variant of dynamic logic. To do this we may simply add the (Finite) or (Infinite) axioms. Or, perhaps we may define a sorted variant of one of the above logics. If the formulae and axioms of this newly developed theory falls into the guarded fragment, it will also be easy to check the decidability and derive a decision procedure from one for guarded matching logic. We believe that this flexibility and modularity, allowing this ease of development of new theories and logics is one of the selling points of matching logic, all with minimal syntactical cruft.

9 Limitations & Future directions

We consider this work a first step into an important research area for matching logic and for the $\mathbb{K}$ framework. There are several directions which would produce additional results of both theoretical interest and practical importance. We discuss a few of them below.

9.1 Extension of the guarded fragment

While the syntactic fragment identified in this work are fairly versatile, there are obvious limitations. For example, nested applications cannot be used in guards. Allowing nested applications is important because it would allow us to encode several important axioms such as associativity, and matching over complex algebraic data types. This seems more like a technical and artificial restriction, and than an inherent limit to decidability. This suspicion is deepened when we look at the difference between the expressivity of guarded applicative matching logic and polyadic matching logic. In applicative matching logic, multi-arity applications are treated as syntactic sugar for binary applications. For the logic as a whole, this does not affect expressivity. However, when considering the guarded fragments, applicative matching logic is significantly less powerful. This suggests a more general principle yet to be identified defines these fragments. Packed logic is a decidable fragment of FOL that allows existentials in guards. This would allow nesting applications up to two deep in matching logic guards. However, we have been unable to find generalizations of packed logic that allow fixedpoints.

Our dependence on translation to loosely guarded FO_{lfp} creates some artificial seeming restrictions on translatable patterns. For example, Requirement (1) is present only because fixedpoint logic does not allow free variables in fixedpoints. The clique-guarded fragment of FO_{lfp} and packed logic (a fragment of FOL) suggest future directions. The clique-guarded fragment is a more general fragment (in terms of expressivity), but is more syntactically restrictive than the loosely-guarded fragment.

The decidability of loosely guarded and clique-guarded logics are based on the bounded tree-width of models of these fragments. There are several other logics whose decidability has been proved based on this property, such as monadic second order logic over graphs of bounded tree-width [Courcelle and Engelfriet 2012], quantifier-free separation logic with recursive definitions [Iosif et al. 2013], and guarded separation logic [Pagel et al. 2020]. In the next few iterations of this work we'd like to identify the principles behind these and expand on the syntactic classes of patterns that are decidable. Proving the bounded-tree width property directly for some fragment of matching logic would also be much more satisfying, and allow us to more easily extend the decidable fragment without being constrained by the syntax and semantics of FO_{lfp}. An ideal outcome of this process would be to identify a large fragment, the bounded tree-width fragment of matching logic, that encompasses many or most of these previous results.

With respect to $\mathbb{K}$'s goals, an important future step would be working with constructors modulo axioms such as associativity and commutativity. $\mathbb{K}$ language definitions consist of a set of conditional transitions over such terms, called rewrite rules. For example, the rule `X:Int / 0 => #Abort` says that division by zero transitions the program to an error state. In order to for the interpreter to take a step for a concrete program, it must match the current program state with the left hand side of this rule. For symbolic execution, where the program state itself may use logical variables, unification is needed. This makes fast and efficient matching and unification vital to $\mathbb{K}$. The "points to" and "separating conjunction" operators of separation logic have many of the properties of constructors. Separation logic with recursive definitions [Iosif et al. 2013] and guarded separation logic [Pagel et al. 2020] are two fragments of this logic that are shown to have the bounded tree-width property. Generalizing the ideas presented there may allow us to extend guarded matching logic to handle matching and unification over constructors modulo axioms. Although not related to the tree-width property, another possible source of inspiration for dealing with constructors modulo axioms is the work on variant-based satisfiability in initial algebras by [Meseguer 2018].

9.2 Combination with semi-decision procedures

Formal verification cannot rely solely on decision procedures. This is because programs and programming languages rely

heavily on domain reasoning over undecidable theories such as the integers. Thus, we must be able to handle best-effort reasoning over these domains, while still handling the decidable portions efficiently.

Traditionally, first-order SMT solvers have done this through combining decision procedures with domain specific reasoning. In particular, the DPLL algorithm [Davis and Putnam 1960] is employed as a core for first-order SMT solvers [Nieuwenhuis et al. 2006]. DPLL is an algorithm for checking the satisfiability of propositional logic formulae. First a propositional "skeleton" of a formulae is created by replacing atoms with variables. The solver then iterates through the (finite) solutions produced by DPLL for this skeleton and tests if any of them are consistent with the atoms, employing domain-specific reasoning as necessary. If one solution is consistent with the atoms, then the original formula is satisfiable.

DPLL may not be used directly for checking the satisfiability of matching logic patterns because propositional logic variables are two-valued, whereas matching logic patterns have a powerset interpretation. We would first need to translate our matching logic patterns to first-order logic. This is not practical because of the large number of quantifiers introduced and the increase in size and complexity of the formula, besides losing the structural information normally preserved in matching logic patterns. Many high-level structural properties important to efficient algorithms would be lost in translation. It may be possible to recover these properties after translation, but the process is likely fragile and expensive.

Instead, we'd like to identify some other decidable fragment that may be used as the core of a future matching logic solver. Guarded matching logic is a candidate we may use to build such a solver. A skeleton may be built by replacing non-guarded quantifiers with uninterpreted symbols. Using an implementation based on [Gradel and Walukiewicz 1999]'s procedure for loosely guarded $\mathrm{FO}_{\mathrm{lfp}}$ we may iterate through models and search for models where the interpretation of these symbols are consistent. This may be done, for example, by passing these more limited parts of the pattern and theory to SMT solvers to perform domain reasoning.

9.3 Implementation

While we have identified a decidable fragment of matching logic, we have not yet implemented a decision procedure yet. As its basis, we may use the procedure described at a high-level in [Gradel and Walukiewicz 1999]. This procedure first builds a possibly infinite tree, through the non-deterministic execution of tableau rules. It then checks that certain properties are met along each infinite path. This is described as an alternating two-way automata over the tableau. To ensure an efficient implementation, much care needs to be taken regarding the representation of this infinite tree, caching common sub-trees while back-tracking, and building only parts of the tableau as needed by the automata. It would be interesting to see if we can build an "incremental" solver – one where we can further constrain a satisfying model while reusing as much as possible from the previous result. This is an important optimization because symbolic execution largely consists of adding path constraints to a program state as transition rules are applied.

10 Related Work

Since the translation of matching logic to $\mathrm{FO}_{\mathrm{lfp}}$ works for most patterns, one may ask, why not just use $\mathrm{FO}_{\mathrm{lfp}}$ as the basis for matching logic? That way we would get the decidability results for that logic for free. This is because we consider notational desugaring within one logic significantly superior to awkward encodings and translations from one logic to another. *Notations* and *axioms* allow capturing logics and formalisms with minimal *representational distance*. It preserves and respects the original syntactic and semantic structures, such as programs, continuations, heaps and stacks, language semantics may be captured in a compact and modular way. For example, the modal μ-calculus $\Diamond\varphi$ is translated to the matching logic pattern (formula) $\Diamond\varphi$. In fact, the formulae of many logics defined in matching logic translate to syntactically identical matching logic patterns. This is in contrast to, for example, fixedpoint logic where additional quantifiers and other techncal infrastructure make working directly with the embedding tedious and heavy.

There are a few other efforts made for automatically checking the validity of matching logic patterns.

As mentioned previously, many of the $\mathbb{K}$'s tools may be thought of as best-effort implementations for checking the validity of certain patterns. The symbolic backend for matching logic [The K Team 2018], which provides a symbolic rewriting engine, a deductive prover and a bounded model checker, is the most general such tool. Again, this is a best-effort implementation for a limited set of theories. For example, it does not attempt any inductive reasoning unless specified as lemmas or circularities. All quantification and other complex logical reasoning is farmed off to the Z3 SMT solver. Our results allow handling both induction and quantifiers in restricted cases.

In [Chen et al. 2020], the authors began the implementation of an automated prover for matching logic, focusing on automating the treatment of fixedpoints. Their prover did not implement a decision procedure, but used a naive depth-first search over proof rules, leading to redundancies and inefficiencies in the proof search. Our approach enables a more systematic search, taking advantage of the tableaux method for loosely guarded $\mathrm{FO}_{\mathrm{lfp}}$. The [Chen et al. 2020] prover was able to handle several theories including LTL, reachability logic, and quantifier-free separation logic with recursive definitions. Of these, an implementation based on

this work would only handle LTL (among others). However, it would be complete for all theories it supports.

11 Conclusion

In this paper, we described a syntactic fragment of matching logic that is decidable. This fragment allows quantification, and so allows the decidability for several theories to be shown by corollary.

References

Hajnal Andréka, István Németi, and Johan van Benthem. 1998. Modal languages and bounded fragments of predicate logic. *Journal of philosophical logic* 27, 3 (1998), 217–274.

Clark Barrett, Aaron Stump, Cesare Tinelli, et al. 2010. The smt-lib standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, England)*, Vol. 13. 14.

Denis Bogdanas and Grigore Roşu. 2015. K-Java: A complete semantics of Java. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. 445–456.

Ashok Chandra and David Harel. 1982. Structure and complexity of relational queries. *Journal of Computer and system Sciences* 25, 1 (1982), 99–128.

Xiaohong Chen, Zhengyao Lin, Minh-Thai Trinh, and Grigore Roşu. 2021a. Towards a Trustworthy Semantics-Based Language Framework via Proof Generation. In *Proceedings of the 33rd International Conference on Computer-Aided Verification*. ACM.

Xiaohong Chen, Dorel Lucanu, and Grigore Roşu. 2021b. Matching logic explained. *Journal of Logical and Algebraic Methods in Programming* 120 (2021), 100638. https://doi.org/10.1016/j.jlamp.2021.100638

Xiaohong Chen and Grigore Roşu. 2019. *Applicative matching logic*. Technical Report http://hdl.handle.net/2142/104616. University of Illinois at Urbana-Champaign.

Xiaohong Chen and Grigore Roşu. 2019. Matching μ-Logic. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 1–13.

Xiaohong Chen, Minh-Thai Trinh, Nishant Rodrigues, Lucas Peña, and Grigore Roşu. 2020. Towards a unified proof framework for automated fixpoint reasoning using matching logic. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 1–29.

Alonzo Church. 1936. An unsolvable problem of elementary number theory. *American journal of mathematics* 58, 2 (1936), 345–363.

Bruno Courcelle and Joost Engelfriet. 2012. *Graph structure and monadic second-order logic: a language-theoretic approach*. Vol. 138. Cambridge University Press.

Andrei Ştefănescu, Daejun Park, Shijiao Yuwen, Yilong Li, and Grigore Roşu. 2016. Semantics-based program verifiers for all languages. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'16)*. ACM, Amsterdam, Netherlands, 74–91.

Sandeep Dasgupta, Daejun Park, Theodoros Kasampalis, Vikram S Adve, and Grigore Roşu. 2019. A complete formal semantics of x86-64 user-level instruction set architecture. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 1133–1148.

Martin Davis and Hilary Putnam. 1960. A computing procedure for quantification theory. *Journal of the ACM (JACM)* 7, 3 (1960), 201–215.

Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

Erich Grädel. 2002. Guarded fixed point logics and the monadic theory of countable trees. *Theoretical Computer Science* 288, 1 (2002), 129–152.

Erich Gradel and Igor Walukiewicz. 1999. Guarded fixed point logic. In *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158)*. IEEE, 45–54.

Yuri Gurevich. 1984. Toward logic tailored for computational complexity. In *Computation and proof theory*. Springer, 175–216.

Chris Hathhorn, Chucky Ellison, and Grigore Roşu. 2015. Defining the undefinedness of C. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 336–345.

D Hilbert and W Ackermann. 1930. Grundzüge der theoretischen Logik. *Bull. Amer. Math. Soc* 36 (1930), 22–25.

Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon Moore, Daejun Park, Yi Zhang,

Andrei Stefanescu, et al. 2018. KEVM: A complete formal semantics of the ethereum virtual machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE, 204–217.

Ian Hodkinson. 2002. Loosely guarded fragment of first-order logic has the finite model property. *Studia Logica* 70, 2 (2002), 205–240.

Radu Iosif, Adam Rogalewicz, and Jiri Simacek. 2013. The Tree Width of Separation Logic with Recursive Definitions. In *Automated Deduction – CADE-24*, Maria Paola Bonacina (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 21–38.

Theodoros Kasampalis, Daejun Park, Zhengyao Lin, Vikram S Adve, and Grigore Roşu. 2021. Language-parametric compiler validation with application to LLVM. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM New York, NY, USA, Virtual, 1004–1019.

José Meseguer. 2018. Variant-based satisfiability in initial algebras. *Science of Computer Programming* 154 (2018), 3–41.

Michael Mortimer. 1975. On languages with two variables. *Mathematical Logic Quarterly* 21, 1 (1975), 135–140.

Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 2006. Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T). *J. ACM* 53, 6 (Nov. 2006), 937–977. https://doi.org/10.1145/1217856.1217859

Damian Niwiński and Igor Walukiewicz. 1996. Games for the μ-calculus. *Theoretical Computer Science* 163, 1 (1996), 99–116. https://doi.org/10.1016/0304-3975(95)00136-0

Jens Pagel, Christoph Matheja, and Florian Zuleger. 2020. A Decision Procedure for Guarded Separation Logic: Complete Entailment Checking for Separation Logic with Inductive Definitions. *arXiv preprint arXiv:2002.01202* (2020).

Daejun Park, Andrei Stefănescu, and Grigore Roşu. 2015. KJS: A complete formal semantics of JavaScript. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 346–356.

Grigore Roşu. 2017a. K—A semantic framework for programming languages and formal analysis tools. In *Dependable Software Systems Engineering*. IOS Press.

Grigore Roşu. 2017b. Matching Logic. *Logical Methods in Computer Science* Volume 13, Issue 4 (Dec. 2017). https://doi.org/10.23638/LMCS-13(4:28)2017

The K Team. 2018. *Haskell Backend*. https://github.com/kframework/kore/

Alan Mathison Turing. 1937. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London mathematical society* 2, 1 (1937), 230–265.

Alasdair Urquhart. 1981. Decidability and the Finite Model Property. *Journal of Philosophical Logic* 10, 3 (1981), 367–370. http://www.jstor.org/stable/30226231

Moshe Y Vardi. 1997. *Why is modal logic so robustly decidable?* Technical Report.

A Equivalence of satisfiability of φ and sat?(φ)

For a matching logic signature, Σ, we define a $\mathrm{FO}_{\mathrm{lfp}}$ signature, $\Sigma_{\mathrm{FO}_{\mathrm{lfp}}}$ with relation symbols R_σ of arity $n+1$ for every symbol $\sigma \in \Sigma$ of arity n.

Given a matching logic model M and an evaluation ρ, we may define a $\Sigma_{\mathrm{FO}_{\mathrm{lfp}}}$ model M', with identical carrier sets. For each symbol, we define the interpretation of the corresponding relation R_σ as: $(x, \bar{a}) \in R_\sigma$ iff $x \in \sigma_M(\bar{a})$ Conversely, given a $\Sigma_{\mathrm{FO}_{\mathrm{lfp}}}$ model we may construct a matching logic Σ-model, M similarly. For each relation R_σ, we define the interpretation of the corresponding symbol σ as: $(x, \bar{a}) \in R_\sigma$ iff $x \in \sigma_M(\bar{a})$

In either case, we show that $a \in |\varphi|_{M,\rho} \iff M', \rho_{[x \mapsto a]} \models \mathsf{matches?}(x, \varphi)$, by structural induction.

$$
\begin{aligned}
& M', \rho_{[x\mapsto a]} \models \mathsf{matches?}(x, \varphi \wedge \psi) \\
\iff & M', \rho_{[x\mapsto a]} \models \mathsf{matches?}(x, \varphi) \wedge \mathsf{matches?}(x, \psi) \\
\iff & M', \rho_{[x\mapsto a]} \models \mathsf{matches?}(x, \varphi) \text{ and } M', \rho_{[x\mapsto a]} \models \mathsf{matches?}(x, \psi) \\
\iff & a \in |\varphi|_{M,\rho} \text{ and } a \in |\psi|_{M,\rho} \\
\iff & a \in |\varphi \wedge \psi|_{M,\rho}
\end{aligned}
$$

$$
\begin{aligned}
& M', \rho_{[x\mapsto a]} \models \mathsf{matches?}(x, \exists y.\, \varphi) \\
\iff & M', \rho_{[x\mapsto a]} \models \exists y.\, \mathsf{matches?}(x, \varphi) \\
\iff & \exists b.\, M', \rho_{[x\mapsto a, y\mapsto b]} \models \mathsf{matches?}(x, \varphi) \\
\iff & \exists b.\, a \in |\varphi|_{M, \rho_{[y\mapsto b]}} \\
\iff & a \in \bigcup_{b \in M} |\varphi|_{M, \rho_{[y\mapsto b]}} \\
\iff & a \in |\exists y.\, \varphi|_{M,\rho}
\end{aligned}
$$

$$
\begin{aligned}
& M', \rho_{[x\mapsto a]} \models \mathsf{matches?}(x, \mu Y.\, \varphi(Y)) \\
\iff & M', \rho_{[x\mapsto a]} \models \left[\mathsf{LFP}_{Y,y}.\, \mathsf{matches?}(y, \varphi(Y))\right](x) \\
\iff & a \in \mathrm{LFP}\left(A \mapsto \left\{a' \in M \mid M, \rho_{[Y\mapsto A, y\mapsto a']} \models \mathsf{matches?}(y, \psi(Y))\right\}\right) \\
\iff & a \in \mathrm{LFP}\left(A \mapsto |\psi(Y)|_{\rho_{[Y\mapsto A]}}\right) \\
\iff & a \in |\mu Y.\, \varphi|_{M,\rho}
\end{aligned}
$$

$$
\begin{aligned}
& M', \rho_{[x\mapsto a]} \models \mathsf{matches?}(x, \sigma(\varphi_1, \ldots, \varphi_n)) \\
\iff & M', \rho_{[x\mapsto a]} \models \exists \bar{y}.\, R_\sigma(x, \bar{y}) \wedge \bigwedge_{1 \le i \le n} \mathsf{matches?}(y_i, \varphi_i) \\
\iff & \exists \bar{b} \in M' \text{ such that } M', \rho_{[x\mapsto a, \bar{y}\mapsto \bar{b}]} \models R_\sigma(x, \bar{y}) \wedge \bigwedge_{1 \le i \le n} \mathsf{matches?}(y_i, \\
\iff & \exists \bar{b} \in M' \text{ such that } M', \rho_{[x\mapsto a, \bar{y}\mapsto \bar{b}]} \models \mathsf{app}(y_1, y_2, x) \text{ and for each } i, \\
& \qquad\qquad M', \rho_{[x\mapsto a, \bar{y}\mapsto \bar{b}]} \models \mathsf{matches?}(y_i, \varphi_i)
\end{aligned}
$$

$\iff \exists \bar{b} \in M'$ such that $a \in \sigma'_M(\bar{b})$ and for each i,
$M', \rho_{[x \mapsto a, \bar{y} \mapsto \bar{b}]} \models \mathsf{matches?}(y_i, \varphi_i)$

$\iff \exists \bar{b} \in M'$ such that $a \in \sigma'_M(\bar{b})$ and for each i,
$M', \rho_{[x \mapsto a, y_i \mapsto b_i]} \models \mathsf{matches?}(y_i, \varphi_i)$
since x and y_j for $j \neq i$ do not occur free in φ_i

$\iff \exists \bar{b} \in M'$ such that $a \in \sigma'_M(\bar{b})$ and for each i,
$b_i \in |\varphi_i|_{M,\rho}$
by induction

$\iff a \in \mathsf{app}_M(|\varphi_1|_{M,\rho}, \ldots, |\varphi_n|_{M,\rho})$

$\iff a \in |\sigma(\varphi_1, \ldots, \varphi_n)|_{M,\rho}$